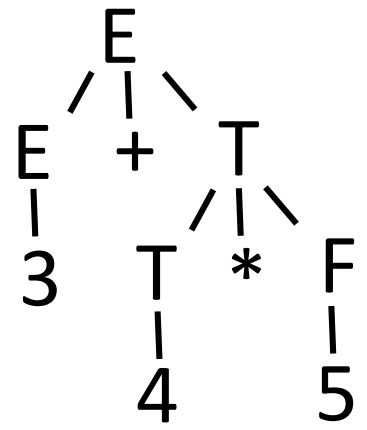
How Grammars Affect Languages

We would like to use grammars for *evaluating* strings in a language. For example, 3+4*5 might have a parse tree such as

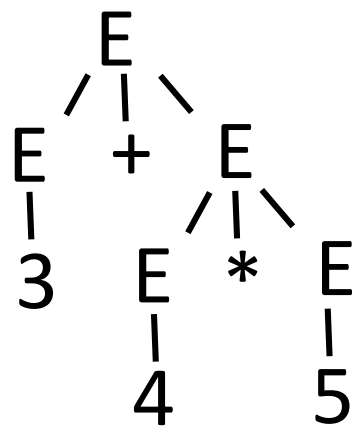


A simple recursive function evaluates this tree to determine the value 23 for 3+4*5.

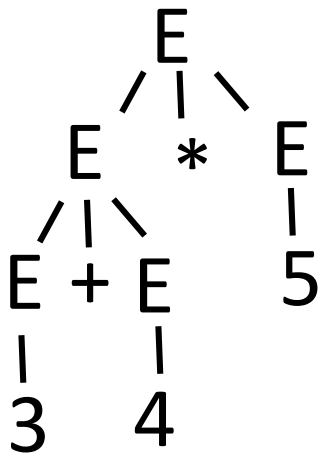
Like every good thing there are potential problems with this. Consider the grammar

E => E+E | E*E | (E) | E digit | digit

This gives two different parse trees for 3+4*5 and those lead to different values for the expression:



Value = 23



Value = 35

A grammar is called *ambiguous* if it produces two different parse trees for the same string. Ambiguity is bad; it prevents us from using parse trees to evaluate strings.

The grammar E => E+E | E*E | (E) | E digit | digit is ambiguous.

The grammar

E => E+T | T
T => T*F | F
F => (E) | G
G => G digit | digit

unambiguously describes the same language.

To show that the grammar

E => E+T | T

T => T*F | F

F => (E) | G

G => G digit | digit

is unambiguous consider two derivations that give different parse trees. There must be a first step in which one derivation uses the rule E => E+T and the other uses E => T => T*F. Let α be the string that both of these derive. If the + occurs in α before the * , then the T in T*F from the second derivation will need to derive +, which it can't. Alternatively, if the * comes in α before the + , then the F term in T*F of the second derivation will need to derive a +, which it can't.

The difference between the grammars

E => E+E | E*E | (E) | E digit | digit

and

E => E+T | T

T => T*F | F

F => (E) | G

G => G digit | digit

is that the latter grammar is hierarchical; operators appear in a specific order within the grammar rules.

We can use such hierarchies to disambiguate grammars.

The problem of determining whether a given grammar is ambiguous is undecidable.

There are languages that are inherently ambiguous -- every grammar for the language is ambiguous. The problem of determining whether a given language is inherently ambiguous is also undecidable.

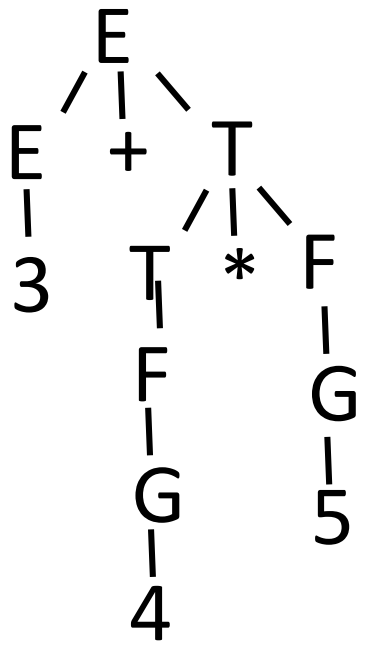
Consider again the string 3+4*5 and its parse trees under the grammars

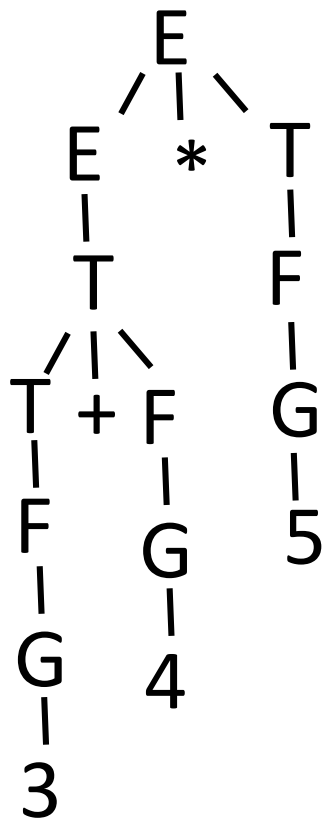
E => E+T | T

T => T*F | F

F => (E) | G

G => G digit | digit





E => E*T | T

T => T+F | F

F => (E) | G

G => G digit | digit

The grammar

E => E+T | T

T => T*F | F

F => (E) | G

G => G digit | digit

has * lower in the hierarchy than + and gives multiplication precedence over addition. The grammar

E => E*T | T

T => T+F | F

F => (E) | G

G => G digit | digit

has + farther down the list of rules and give addition precedence over multiplication

In general, with hierarchical grammars the farther down in the list of rules an operator appears, the greater its precedence will be.

There is another property grammars give languages. Consider the two grammars

E => E-T | T
T => (E) | G
G => G digit | digit

and

E => T-E | T
T => (E) | G
G => G digit | digit

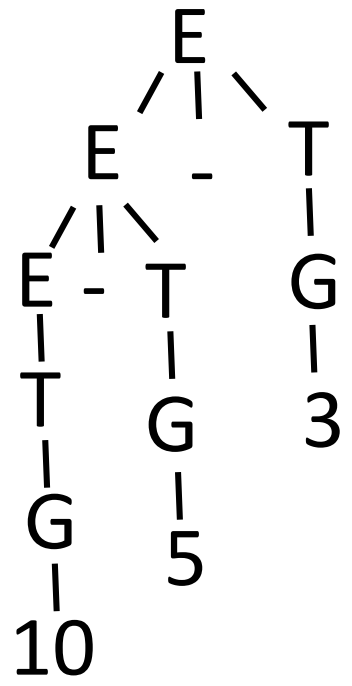
These differ only in whether the E-rule is left- or right-recursive.

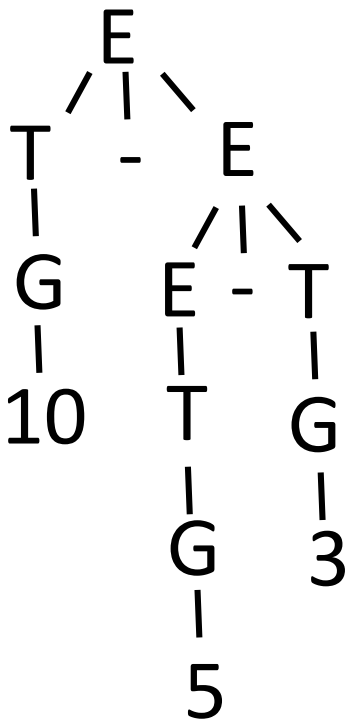
We use these grammars to parse the string 10-5-3

E => E-T | T

T => (E) | G

G => G digit | digit





E => T-E | T

T => (E) | G

G => G digit | digit

Note that the left parse tree evaluates to 2 while the right one evaluates to 8.

In general, left-recursive rules lead to left-associative operators while right-recursive rules lead to right-associative operators. We want most of the standard arithmetic operators to be left-associative, so we use rules such as E => E+T.

Ambiguity is bad but it can't always be avoided. Some languages are *inherently* ambiguous, in that every grammar for the language is ambiguous. Here is an example, though we don't prove the inherent ambiguity:

$\mathcal{L} = \{a^n b^n c^m d^m \mid n,m >= 1\} \cup \{a^n b^m c^m d^n \mid n,m >= 1\}$

(Either the a's and b's match and the c's and d's match
or else the a's and d's match and the b's and c's match.
Note that $a^n b^n c^n d^n$ satisfies both conditions.)

Here is one grammar for this language:

S ==> AB | C
A ==> aAb | ab
B ==> cBd | cd
C ==> aCd | aDd
D ==> bDc | bc

Here are two derivations for the string abcd:

S => AB
  => abB
  => abcd

S => C
  => aDd
  => abcd

Of course, showing that one particular grammar for a language is ambiguous doesn't show that all grammars for it are ambiguous but in this case that is true.